An Open Source Solar Power Forecasting Tool Using PVLIB-Python

William F. Holmgren*, Derek G. Groenendyk†

*Department of Atmospheric Sciences, University of Arizona, Tucson, AZ, 85721, United States
†Department of Hydrology, University of Arizona, Tucson, AZ, 85721, United States

***Abstract*—We describe an open-source PV power forecasting tool based on the PVLIB-Python library. The tool allows users to easily retrieve standardized weather forecast data relevant to PV power modeling from NOAA/NCEP/NWS models including the GFS, NAM, RAP, HRRR, and the NDFD. A PV power forecast can then be obtained using the weather data as inputs to the comprehensive modeling capabilities of PVLIB-Python. Standardized, open source, reference implementations of forecast methods using publicly available data may help advance the state-of-the-art of solar power forecasting.**

***Index Terms*—forecasting, performance modeling, PV modeling, software**

I. Introduction

Solar power forecast methods continue to be developed at a rapid pace, a necessary development for the integration of substantial amounts of renewable energy in the electrical grid [1]. A large number of solar forecasting algorithms have been developed that make use of weather models developed and operated by the national weather services or consortiums of many countries. While these weather models have seen significant improvements in accuracy over the last several decades, solar power forecasting places new requirements on these models and many models fail in some basic ways. For example, many weather models do not use an accurate solar position equation (the equation of time) and do not account for aerosols in their radiative transfer algorithms [2]. Researchers and forecast providers benchmark their new and possibly proprietary weather models or machine learning algorithms on these weather models. However, the calculation of PV output, let alone irradiance, from weather models can be done in many different ways. For example, should the model's incoming shortwave radiation (if available) or the model's cloud fraction be used? Or perhaps a blend of the two, potentially conditioned on another model variable, could be used. Larson et. al. recently evaluated the day-ahead accuracy of solar power forecasts in the Southwestern United States and concluded that researchers should shift their focus from irradiance to solar power forecasts [3]. Even in our own experience in solar and wind power forecasting at the University of Arizona [4], we struggle to fairly, comprehensively, and transparently answer the basic question of "How much better is your forecast?".

We propose that both public and private solar power forecasters will benefit from standardized, open source, reference implementations of forecast methods that use publicly available data. In this paper, we will demonstrate a tool for the open source PVLIB-Python library that allows for simple access to publicly available weather forecast data that is readily converted into a PV power forecast. We will outline the basic structure of the tool, describe some of the design challenges, and finally walk through a simple example of how to use the tool to create a solar power forecast. As of this writing, the tool is available on the forecast branch of the PVLIB Python git repository [5] with documentation on readthedocs.io [6].

PVLIB-Python is an BSD 3-clause open source library for photovoltaic modeling and analysis for the Python programming language [7], [8]. The library is based on the PVLIB Matlab library originally developed at Sandia National Laboratories [9]. PVLIB-Python is developed on GitHub and uses modern development practices such as version control, continuous integration testing, and automated documentation builds [10]. The use of these tools in PVLIB-Python is described in more detail in [8]. The Python language, along with the open source Scientific Python (or PyData) stack of libraries, provides a compelling platform for data collection, analysis, network interfaces, web server integration, and much more. Python has also been identified by Unidata, an essential developer and maintainer of geosciences software, as a key technology for geosciences [11]. Thus, PVLIB and Python are natural choices for developing an open source tool that combines weather forecasting and PV modeling.

II. Forecasting with PVLIB-Python

Creating a PV power forecast in PVLIB-Python can be broken into two steps: accessing weather model data and converting the weather model data into a power forecast.

A. Accessing Weather Model Data in PVLIB-Python with Siphon

Unidata maintains and develops the Siphon library [12], a tool to designed to make it easy to programmatically access and download geosciences data in Python. The Siphon library provides access to, among others, forecasts from the Global Forecast System (GFS), North American Model (NAM), High Resolution Rapid Refresh (HRRR), Rapid Refresh (RAP), and National Digital Forecast Database (NDFD) on a Unidata THREDDS server. Unfortunately, many of these models use different names to describe the same quantity (or a very similar one), and not all variables are present in all models. For example, on the THREDDS server, the GFS has a field named 'Total_cloud_cover_entire_atmosphere_Mixed_intervals_Average', while the RAP has a field named 'Total_cloud_cover_entire_atmosphere_single_layer',

and a similar field in the HRRR is named 'Total_cloud_cover_entire_atmosphere'.

PVLIB-Python aims to simplify the access of the model fields relevant for solar power forecasts. All models accessed via PVLIB-Python are returned with uniform field names: temperature, wind_speed, total_clouds, low_clouds, mid_clouds, high_clouds, dni, dhi, ghi. To accomplish this, we created an object-oriented framework in which each weather model is represented by a class that inherits from a parent `ForecastModel` class. The parent `ForecastModel` class contains the common code for accessing and parsing the data using Siphon, while the child model-specific classes (`GFS`, `HRRR`), etc.) contain the code necessary to map and process that specific model's data to the standardized fields.

The code below demonstrates how to access and plot forecast data using PVLIB-Python. This code accesses 7 days of forecast data from the most recent GFS model and creates the plot shown in Figure 1.

```
# import forecast models
from pvlib.forecast import GFS, NAM, NDFD, HRRR

# specify location
latitude = 32.2
longitude = -110.9
tz = 'US/Arizona'

# specify time range of start of today to 7 days
start = pd.Timestamp(datetime.date.today(), tz=tz)
end = start + timedelta(days=7)

# GFS model, defaults to 0.5 degree resolution
# 0.25 deg available
model = GFS()

# retrieve data. returns pandas.DataFrame object
data = model.get_processed_data(latitude, longitude,
                                start, end)

# plot cloud cover percentages
cloud_vars = ['total_clouds', 'low_clouds',
              'mid_clouds', 'high_clouds']
data[cloud_vars].plot()
plt.ylabel('Cloud cover %')
plt.xlabel('Forecast Time ({})'.format(tz))
plt.title('GFS 0.5 deg forecast for lat={}, lon={}'
          .format(latitude, longitude))
plt.legend()
```

The code works as follows. After the basic imports and location specifications, we created an object, `model`, corresponding to the GFS model, and then we called a method, `model.get_processed_data`, to retrieve the data from Unidata's THREDDS server using the Siphon library and process it into the relevant variables for PV power. This method call returns a Pandas DataFrame, a convenient and powerful data structure that is essentially a labeled two dimensional array. Finally, we plotted a subset of the data, the cloud cover forecasts.

The `model.get_processed_data` method is a convenience method that combines two methods: `model.get_data` and `model.process_data`. We emphasize that the `get_data` and `process_data` methods can be called independently of `get_processed_data` and that this creates a powerful yet flexible framework in which to experiment with new data retrieval and processing algorithms. The parent `ForecastModel` implements the `get_data` method and a basic `process_data` method. Each child model implements its own `process_data` method that calls the parent model's `process_data` method and additional methods to convert cloud cover to irradiance, normalize surface winds, and convert temperatures, if necessary. For example, the code below shows the abbreviated method definitions for the GFS and NAM models that have slightly different processing steps.

```
# in the GFS class
def process_data(
    self, data, cloud_cover='total_clouds', **kwargs):
    """
    Defines the steps needed to convert raw forecast
    data into processed forecast data.

    Parameters
    ----------
    data: DataFrame
        Raw forecast data
    cloud_cover: str
        The type of cloud cover used to infer
        the irradiance.

    Returns
    -------
    data: DataFrame
        Processed forecast data.
    """
    data = \
        super(GFS, self).process_data(data, **kwargs)
    data['temperature'] = \
        self.kelvin_to_celsius(data['temperature'])
    data['wind_speed'] = self.uv_to_speed(data)
    data = data.join(
        self.cloud_cover_to_irradiance(
            data[cloud_cover]),
        how='outer')
    return data.ix[:, self.output_variables]

# in the HRRR class
def process_data(
    self, data, cloud_cover='total_clouds', **kwargs):

    data = \
        super(HRRR, self).process_data(data, **kwargs)
    data['temperature'] = \
        self.isobaric_to_ambient_temperature(data)
    data['temperature'] = \
        self.kelvin_to_celsius(data['temperature'])
    data['wind_speed'] = self.gust_to_speed(data)
    data = data.join(
        self.cloud_cover_to_irradiance(
            data[cloud_cover]),
        how='outer')
    return data.ix[:, self.output_variables]
```

PVLIB-Python currently uses the Liu-Jordan model [13] to convert total cloud cover forecasts to irradiance forecasts, though it is simple to implement new models and provide additional options. Figure 2 shows the result of the cloud cover to irradiance conversion. Note that the GFS data shown here has a time resolution of 3-hours, thus the default irradiance forecasts also have a 3 hour time resolution. However, it is straightforward to interpolate the cloud cover forecasts onto a higher resolution time domain, and then recalculate the

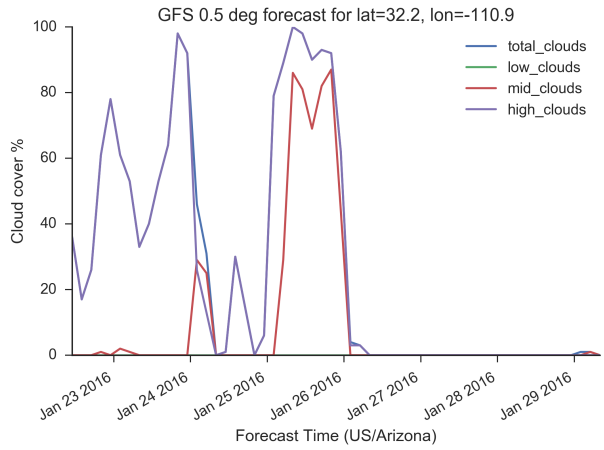

Fig. 1. Seven days of cloud cover forecasts from the GFS model for Tucson, Arizona. The GFS model provides clouds total cloud cover (blue), low cloud cover (green), mid cloud cover (red), and high cloud cover (purple). Multiple cloud levels can be used to create a more accurate irradiance forecast. The figure is generated using only the code shown in the text.

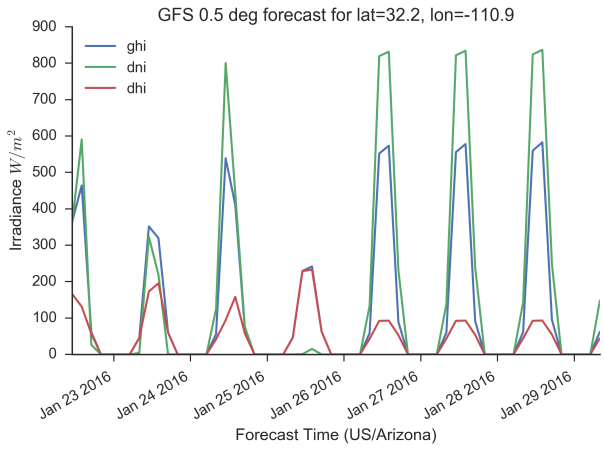

Fig. 2. Seven days of GHI (blue), DNI (green), and DHI (red) forecasts derived from the GFS model for Tucson, Arizona. PVLIB-Python currently uses the Liu-Jordan model to convert cloud cover to irradiance.

irradiance. An example of this procedure is shown in the online documentation [6]. We encourage users to become familiar with the source code so that they may understand what, exactly, the algorithms do and we reiterate that the open source, permissively licensed, and accessible code enables users to customize the model processing to their liking.

B. PV Power Forecasting

The standardized weather model data can now be easily integrated with PVLIB-Python's power modeling tools. For a detailed explanation of the power modeling steps and examples, we refer to reader to PVLIB-Python's online documentation and IPython Notebook tutorials [6]. In brief, the plane of array irradiance (POA) is calculated from the GHI, DNI, and DHI using a built-in solar position calculator and transposition model. Then, a module and inverter are specified by name from

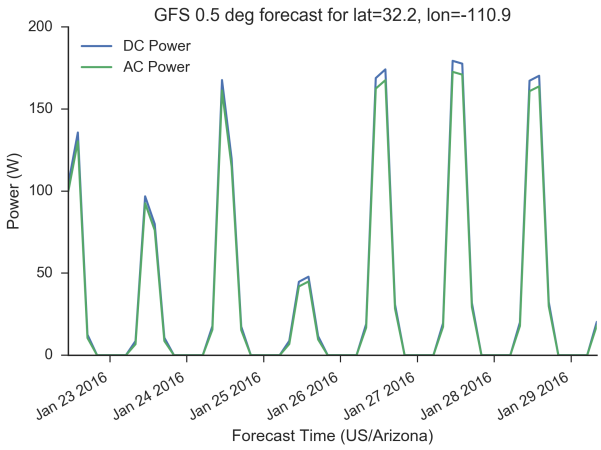

Fig. 3. Seven days of DC (blue) and AC (green) power forecasts derived from the GFS model for Tucson, Arizona.

an online database. The forecasts of ambient temperature and wind speed, plus the module parameters, are used to calculate PV cell and module temperatures. DC power is calculated using the Sandia Array Performance Model or the single diode model, using the forecast POA irradiance, forecast module temperature, and reference module specifications. Finally, the Sandia Inverter Model is applied to calculate the AC power forecast. The online documentation demonstrates that many of these steps can be automated via a `ModelChain` object [6]. Figure 3 shows an example of the DC and AC power forecast for a single module and a small inverter using the same set of GFS forecast data shown above.

III. Conclusion

We described a new module for the PVLIB-Python library that enables users to easily access weather forecasts and process them into PV power forecasts. The tool creates a standard set of data for PV modeling from the sparse and mixed types of data provided from weather models. The code is open source, written in an accessible language, and uses online version control, automated testing, and documentation services. We believe that it is an important step towards the standardization of PV power forecast methods, simplification of the use of weather forecast data, and fair and transparent PV power forecast model performance evaluation.

Finally, we encourage readers to download the project, use the code, and propose improvements and new directions using our GitHub page [10].

Acknowledgments

The authors gratefully acknowledge Sandia National Laboratories for the initial development of PVLIB-MATLAB and PVLIB-Python and the ongoing contributions of many others to the project. A list of PVLIB-Python contributors may be found on the GitHub repository [10] and in the online documentation [14]. The authors also gratefully acknowledge Unidata, the Siphon developers, and the NCEP, NWS, and

other NOAA personnel that create, develop, and provide access to the weather models discussed above. WFH thanks the Department of Energy (DOE) Office of Energy Efficiency and Renewable Energy (EERE) Postdoctoral Research Award for support.

REFERENCES

[1] J. Kleissl, *Solar Energy Forecasting and Resource Assessment*. Oxford, UK: Elsevier, 2013.

[2] P. A. Jimenez, J. P. Hacker, J. Dudhia, S. Ellen Haupt, J. A. Ruiz-Arias, C. A. Gueymard, G. Thompson, T. Eidhammer, and A. Deng, "Wrf-solar: An augmented nwp model for solar power prediction. model description and clear sky assessment," *Bulletin of the American Meteorological Society*, 2015/12/14 2015. [Online]. Available: http://dx.doi.org/10.1175/BAMS-D-14-00279.1

[3] D. P. Larson, L. Nonnenmacher, and C. F. M. Coimbra, "Day-ahead forecasting of solar power output from photovoltaic plants in the american southwest," *Renewable Energy*, vol. 91, pp. 11–20, 6 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0960148116300398

[4] W. F. Holmgren, A. T. Lorenzo, M. Leuthold, C. K. Kim, A. D. Cronin, and E. A. Betterton, "An operational, real-time forecasting system for 250 mw of pv power using nwp, satellite, and dg production data," *PVSC Proceedings*, 2014.

[5] PVLIB Python contributors. pvlib/pvlib-python. [Online]. Available: http://pvlib-python.readthedocs.io/en/forecast/forecasts.html

[6] W. F. Holmgren and D. G. Groenendyk. pvlib-python forecast branch documentation. [Online]. Available: http://pvlib-python.readthedocs.io/en/forecast/forecasts.html

[7] R. W. Andrews, J. S. Stein, C. Hansen, and D. Riley, "Introduction to the open source PV LIB for Python Photovoltaic system modelling package," in *40th IEEE Photovoltaic Specialist Conference*, 2014.

[8] W. F. Holmgren, R. W. Andrews, A. T. Lorenzo, and J. S. Stein, "Pvlib python 2015," in *42th IEEE Photovoltaic Specialist Conference*, 2015.

[9] J. S. Stein, "The photovoltaic performance modeling collaborative (PVPMC)," in *Photovoltaic Specialists Conference*, 2012.

[10] PVLIB Python contributors. pvlib/pvlib-python. [Online]. Available: https://github.com/pvlib/pvlib-python

[11] Unidata. Unidata projects: Python. [Online]. Available: http://www.unidata.ucar.edu/projects/index.html#python

[12] ——. Siphon. [Online]. Available: https://github.com/Unidata/siphon

[13] B. Y. H. Liu and R. C. Jordan, "The interrelationship and characteristic distribution of direct, diffuse and total solar radiation," *Solar Energy*, vol. 4, no. 3, pp. 1–19, 7 1960. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0038092X60900621

[14] PVLIB Python contributors. pvlib-python documentation. [Online]. Available: https://pvlib-python.readthedocs.io